

User's guide for the ROC-HJ* solver: Finite Differences and Semi-Lagrangian methods

November 2014

Version 2.0[†]

Current developers : O. Bokanowski, H. Zidani, J. Zhao, A. Desilles.

1 Problem solved

This is a c++ MPI/OpenMP library for solving d -dimensional Hamilton-Jacobi-Bellman equations by finite difference methods, or semi-lagrangian methods. First order and second order HJ equations time-dependent or steady equations can be solved. More precisely, the equations considered are of the following type.

1.1 Evolutionary case

The problem is to find $u = u(t, x)$ solution of

$$\begin{cases} \frac{\partial u}{\partial t} + \lambda(x)u(x) + H(t, x, u(x), \nabla u) = 0, & x \in \Omega, \quad t \in [0, T], \\ u(t, x) = g_{bord}(t, x), & x \in \partial\Omega, \quad t \in [0, T], \\ u(0, x) = u_0(x), & x \in \Omega \end{cases} \quad (1)$$

where Ω is a domain of $\mathbb{R}^d$ of the form $\prod_{i=1}^d [a_i, b_i]$, and g_{bord} and u_0 are given functions. (other boundary conditions are possible). The function $H(t, x, p)$ can be defined either directly or as follows¹

$$H(t, x, \nabla u) := \max_{a \in \mathcal{A}} \left(-f(t, x, a) \cdot \nabla u - \ell(t, x, a) \right), \quad (2)$$

*Reachability, Optimal Control, and Hamilton-Jacobi equations

[†]Including a SL solver for second order HJ equations, see Section 7.

1. $\max_{a \in \mathcal{A}}(\dots)$ can be also $\min_{a \in \mathcal{A}}(\dots)$

where A is a set of controls, of the form $\prod_{i=1}^{n_A} [\alpha_i, \beta_i]$, or ²

$$H(t, x, \nabla u) := \max_{a \in \mathcal{A}} \min_{b \in \mathcal{B}} \left(-f(t, x, a, b) \cdot \nabla u - \ell(t, x, a, b) \right), \quad (3)$$

where $\mathcal{A}$ and $\mathcal{B}$ are control sets of the form $\prod_{i=1}^{n_A} [\alpha_i^A, \beta_i^A]$ and $\prod_{j=1}^{n_B} [\alpha_j^B, \beta_j^B]$.

Some second order equations can also be treated, of the following type

$$0 = \frac{\partial u}{\partial t} + \lambda(x)u + \quad (4a)$$

$$\begin{aligned} & \max_{a \in \mathcal{A}} \left\{ -\ell(t, x, a) + r(t, x, a)u - b(t, x, a) \cdot \nabla u - \frac{1}{2} \text{Tr}(\sigma(t, x, a)\sigma(t, x, a)^T D^2 u) \right\} \\ & t \in [0, T], \quad x \in \mathbb{R}^d, \\ & u(0, x) = u_0(x), \quad x \in \mathbb{R}^d \end{aligned} \quad (4b)$$

where $\mathcal{A}$ is some non empty compact subset of $\mathbb{R}^m$ ($m \geq 1$), (of the form $\prod_{i=1}^{n_A} [\alpha_i, \beta_i]$), $b(t, x, a)$ is a vector of $\mathbb{R}^d$, $r(t, x, a)$ and $\ell(t, x, a)$ are real-valued, and $\sigma(t, x, a)$ is a $d \times p$ real matrix (for some $p \geq 1$). This problem is linked to the computation of the value function of stochastic optimal control problems.

1.2 Steady equations

The problem is to find $u = u(x)$ solution of

$$\lambda(x)u(x) + H(x, u(x), \nabla u(x)) = 0, \quad x \in \Omega, \quad (5)$$

with H given directly or in the form of (2) (or as in (4a) for second order equations), the function $\lambda(x)$ should be strictly positive, Ω an hyperrectangle of $\mathbb{R}^d$ of the form $\prod_{i=1}^d [a_i, b_i]$. Equation (5) is complemented by boundary conditions as follows :

$$u(x) = g_{bord}(x), \quad x \in \partial\Omega.$$

It is possible also to solve (5) in a subdomain $\mathcal{C}$. In that case the set $\mathcal{C}$ should be defined such that $\mathcal{C} := \{x, g_{domain}(x) < 0\}$ (equivalently, $\Omega \setminus \mathcal{C} = \{x \in \Omega, g_{domain}(x) \geq 0\}$), and the boundary conditions at the border $\partial\mathcal{C}$ should be of Dirichlet type, fixed by $u_0(x)$:

$$u(x) = u_0(x), \quad x \in \partial\mathcal{C}.$$

In general this equation is solved by using an iterative procedure ("value iteration"), where the part $\lambda(x)u(x)$ is treated explicitly (for a FD method) or implicitly (for a SL method). ³

2. $\max_{a \in \mathcal{A}} \min_{b \in \mathcal{B}}(\dots)$ can be also $\min_{a \in \mathcal{A}} \max_{b \in \mathcal{B}}(\dots)$

3. For steady equations, the scheme is based on the following iteration on $n \geq 0$, for a fixed $\Delta t > 0$:

$$\frac{u^{n+1}(x) - u^n(x)}{\Delta t} + \lambda(x)u^n(x) + h(x, u^n(x), Du^n(x)) = 0, x \in \mathcal{C},$$

2 Compilation and execution (under GCC and CMake)

In order to make the program works, the `c++` compiler GCC and the build system CMake are proposed to help the compilation process. The source files are typically in the directory `src/`. User's data files are typically in the directory `data/`.

For deeper informations, we redirect the user to the following links :

<http://gcc.gnu.org/install/>

<http://www.cmake.org/cmake/resources/software.html>

Building the Makefile : `cmake .` (do not forget the "dot")

Compilation : `make` (source `.cpp` files are generally in the directory `src/`)

Options : `make clean` (cleaning some `.o` files), `make cleanall`.

Other : `./clean` to first erase all unnecessary files (can be done before the "`cmake .`" command.)

Sequential code Execution (basic) :

`./exe`

Execution (with options)

`./exe -nn NN -nc NC`

Options :

- option `-nn NN` : number of mesh points per dimension
- option `-nc NC` : number of controls per command's dimension.

where $h(x, u^n(x), Du^n(x))$ corresponds to some numerical approximation of $H(x, u^n(x), \nabla u^n(x))$. Iterations are stopped when $\|u^{n+1} - u^n\|_{L^\infty}$ is smaller than a given threshold. Therefore we obtain the following recursion

$$u^{n+1}(x) = u^n(x) - \Delta t \left(\lambda(x)u^n(x) + h(x, u^n(x), Du^n(x)) \right) = 0, \quad x \in \mathcal{C}.$$

For the SL method, the iterative scheme is based on an implicit treatment of the $\lambda(x)u(x)$ term, and becomes, in the case of (2) :

$$u^{n+1}(x) = \min_a \frac{1}{1 + \lambda(x)\Delta t} \left([u^n](x + \Delta t f(t, x, a)) + \ell(t, x, a) \right) = 0, \quad x \in \mathcal{C}.$$

Here $[u^n]$ denotes the Q_1 interpolation of u^n on the spatial grid mesh.

OpenMP Some options run with OpenMP. Execution :

```
./exe -nt nbth
```

where `nbth` is the number of threads (typically try `nbth=2` or `nbth=4`).

Parallel code (Parallel MPI version only)

Execution : `mpirun -n MPI_PROCS ./exe -nn NN -nc NC -nt OMP_THREADS -nd MPI_GRID_DIM`
where

- `MPI_PROCS` : number of MPI processors (default = 1)
- `NN` : number of mesh points per dimension
- `NC` : number of controls per command's dimension
- `OMP_THREADS` : number of OpenMP threads (default = 1)
- `MPI_GRID_DIM` : dimension of the MPI mesh grid decomposition (2 or 3) (default = 2).

Example 1 : `mpirun -n 2 ./exe -nn 200` will execute the program with 2 MPI processors and 200 mesh points per dimension.

Example 2 : `mpirun -n 64 ./exe -nn 500 -nc 10 -nt 4 -nd 2` will execute the program with `MPI_PROCS=64` MPI processors, `NN=500` points per dimension, `NC=10` controls per command's dimension, `OMP_THREADS=4` OpenMP threads, and `MPI_GRID_DIM=2`, the dimension of the MPI mesh grid decomposition.

Example 3 : `mpirun -n 1 ./exe -nn 500 -nc 10 -nt 4`.

The OpenMP parallelization is working for Semi Lagrangian and Finite Difference methods. When using only this method, one can indicate only one process for the `mpirun` command (for instance `mpirun -n 1 ./exe -nt 4` for using 4 threads) or, equivalently, use `./exe -nt 4`.

3 User's parameters

The file `data/data_simulation.h` is the main user's input file and contains the parameters that define the equation to be solved. It can include an other `data_XXX.h` file (see basic examples `data_basicmodel.h`, `data_FD_2d_ex1_basic.h`). It contains also parameters relative to the output during the execution saving data and data post-treatment (see paragraph "Output parameters").

- `NAME` : name of the problem
- `DIM` : dimension d of the problem

Choice of the solver

- `METHOD` $\in \{\text{MDF}, \text{MSL}\}$:

MDF : Finite Difference Method

MSL : Semi-Lagrangian Method

Stopping criteria

- T : terminal time
- MAX_ITERATION : to stop the program when this maximum number of iterations is reached.
- EPSILON : for stopping criteria. If set to 0, do nothing. If set to some positive value ϵ , then the program will stop at iteration $n + 1$ as soon as

$$\|v^{n+1} - v^n\|_{L^\infty} := \max_i |v_i^{n+1} - v_i^n| \leq \epsilon.$$

(the L^∞ error between two successive steps is smaller than ϵ).

Finite Difference Method This method is used when METHOD=MDF, and it utilizes :

- COMMANDS $\in \{0, 1, 2\}$:

This decides how we define the Hamiltonian function $H(x, p)$ (using the "commands" or not).

1 : the commands are used in the definition of the numerical hamiltonian, as in (2). The program will use a numerical hamiltonian function corresponding to a finite difference approximation of $H(x, \nabla u) = \max_{\alpha} (-f(x, \alpha) \cdot \nabla u - \ell(x, \alpha))$. Examples are given in `data_basicmodel.h` (rotation), or `data_FD_2d_ex1_basic.h` (eikonal equation).

2 : the commands are used in the definition of the numerical hamiltonian, as in (3). The program will use a numerical hamiltonian function corresponding to a finite difference approximation of (for instance) $H(x, \nabla u) = \max_{\alpha} \min_{\beta} (-f(x, \alpha, \beta) \cdot \nabla u - \ell(x, \alpha, \beta))$.

0 : an `Hnum` function must be directly defined by the user ; it has to be a numerical hamiltonian corresponding to $H(x, p)$. Examples are given in `data_advancedmodel.h`, `data_FD_2d_ex1_advanced.h`.

If COMMANDS=1, we need also to define

- function `dynamics` : $f(x, \alpha)$
- function `distributed_cost` : $\ell(x, \alpha)$
- a set of commands corresponding to the discretization of a cube $\prod_{i=1, n_c} [\alpha_i, \beta_i]$, defined by `cDIM` (dimension n_c of the set of controls), `NCD[cDIM]` (number of commands per direction) `UMIN[cDIM]`, `UMAX[cDIM]` (min and max values α_i and β_i of the commands in each direction).

If COMMANDS=2, we need to define

- function `dynamics2` : $f(x, a, b, t)$
- function `distributed_cost2` : $\ell(x, a, b, t)$
- two sets of commands $\mathcal{A}$ and $\mathcal{B}$, corresponding to the discretization of $\mathcal{A} \equiv \prod_{i=1, n_A} [\alpha_i^A, \beta_i^A]$ and $\mathcal{B} \equiv \prod_{i=1, n_B} [\alpha_i^B, \beta_i^B]$. The parameters `cDIM`, `cDIM2` will correspond to n_A and n_B , the dimension of each set of controls), `NCD[cDIM]` is the number of commands per direction, and `UMIN[cDIM]`, `UMAX[cDIM]` are the min and max values of α_i and β_i of the commands in each direction.

If `COMMANDS=0`, we need to define

- function `Hnum` : a first order numerical Hamiltonian.
- function `compute_Hconst`. This function must define d constants which are bounds for $\left| \frac{\partial H}{\partial p_i}(x, p, t) \right|$ ($i = 1, \dots, d$), for $x \in \Omega$ and for possible gradient values p and time t . This is then used to set the time step Δt in order to satisfy a CFL condition. These constants may also be used in the function `Hnum`.
(The user may also define directly the constants $c_i = \text{Hconst}[i]$ and initialize the previous function accordingly.)

Then, in all cases, we have also to set the scheme discretisation parameters :

- `TYPE_SCHEME` $\in \{\text{LF}, \text{ENO2}\}$: type of spacial discretization
`LF` : Lax-Friedrich scheme (first order scheme)
`ENO2` : ENO scheme of second order to approximate the derivatives ∇u
- `TYPE_RK` $\in \{\text{RK1}, \text{RK2}, \text{RK3}\}$: Time discretization by a Runge-Kutta method of order 1, 2 or 3.
`RK1` : RK method of order 1
`RK2` : RK method of order 2
`RK3` : RK method of order 3

Semi-Lagrangian Method This method is used when `METHOD=MSL`. It assumes that :

$$H(x, \nabla u) \equiv \max_{\alpha} (-f(x, \alpha) \cdot \nabla u - \ell(x, \alpha)).$$

Then it needs to define :

- function `dynamics` : dynamics $f(x, \alpha)$
- function `distributed_cost` : cost function $\ell(x, \alpha)$
- `TYPE_SCHEME` $\in \{\text{STA}, \text{EVO}\}$:
`STA` : stationnary case, for solving (5). In this case, the scheme is based on an iterative procedure.
`EVO` : dynamic case, for solving (1)
`ORDER` : this value is set to 1 for solving first order equations of type (1) (see other section below for second order HJB equations).

- `TYPE_STA_LOOP` $\in \{\text{NORMAL}, \text{SPECIAL}\}$: Mesh loop order during mainloop for the stationary case.
 - `NORMAL` : normal ordering loop
 - `SPECIAL` : special ordering loop (makes 2^d loops at each iteration, modifying the current values of the data $\mathbf{v}$ during each loop)
- `TYPE_RK` $\in \{\text{RK1_EULER}, \text{RK2_HEUN}, \text{RK2_PM}\}$:
 - `RK1\_EULER` : RK1 Euler scheme
 - `RK2\_HEUN` : RK2 Heun scheme
 - `RK2\_PM` : RK2 Mid-point scheme
- `INTERPOLATION` $\in \{\text{BILINEAR}, \text{PRECOMPBL}, \text{DIRPERDIR}\}$:
 - `BILINEAR` : (default value) bilinear interpolation ($Q1$ interpolation)
 - `PRECOMPBL` : precompute the interpolation coefficients (to use with tiny mesh sizes)
 - `DIRPERDIR` : another interpolation method
- `COMPLEMENTS` :
 - `P\_INTERMEDIATE` : number of intermediate timesteps to go from t_n to $t_n + \Delta t$ for computing a characteristic for a given RK method.
 - `ORDER` $\in \{1, 2\}$: decide treatment for first order HJ equation or for second order HJ equation.

Discretization parameters

- `ND[DIM]` : tab containing the size mesh in each direction (cartesian mesh)
- `MESH` $\in \{0, 1\}$: default is 1. It utilizes `ND[i]+1` points in direction i . (If `MESH=0` the mesh points are at the center of the mesh cells, and there are `ND[i]` points in direction i)
- `DT` : the time step used for the solver, for the evolutive equation.
 - For the finite difference approach, if `DT=0`, then time step `DT` is computed so that the CFL condition be satisfied, that is, such that

$$\text{DT} * (\text{Hconst}[0]/\text{dx}[0] + \text{Hconst}[1]/\text{dx}[1] + \dots) \leq \text{CFL}$$
 where the CFL number belongs to $[0, 1]$. Otherwise, if `DT>0`, then the value `DT` is used for the time step.
 - For the semi-lagrangian approach and for the stationary equation, the parameter h in the iterative procedure is also set to `DT`

Other parameters for the definition of the problem

The parameters are :

- `XMIN[DIM]` , `XMAX[DIM]` : the boundary (a_i, b_i) of the domain in each direction
- `PERIODIC[DIM]` : each component set the periodicity for each direction
 - 1 : direction is periodic
 - 0 : direction is not periodic
- `BOUNDARY` $\in \{0, 1, 2\}$: If set to 0, no boundary treatment. If set to 1 : Utilizes Dirichlet

boundary conditions for each direction $\mathbf{d}$ such that `PERIODIC[d]=0`. The boundary value is given in the user-defined function `g_bord`. If set to 2 : Utilizes a special $v_{x_i x_i} = 0$ boundary condition for each direction $\mathbf{d}$ such that `PERIODIC[d]=0` (note that this special approximation can be unstable).

- function `v0` : the initial data
- function `Vex` : if known, the user can put the exact solution of the problem here.
- integer `EXTERNAL_v0` $\in \{0, 1\}$: if 0 initialize with function `v0`, otherwise initialize with last `VF.dat` values (located in `OUTPUT/VF.dat`)⁴
- `COMPUTE_IN_SUBDOMAIN` $\in \{0, 1\}$: determines if subdomain computations should be done, so that the evaluation of $u^{n+1}(x)$ is done only at grid mesh point x such that $g_{domain}(x) < 0$.
- function `g_domain(x)` : function used to define the subdomain.

Obstacle terms Instead of solving $u_t + H(t, x, u, \nabla u) = 0$, the solver can also treat HJ obstacle equations such as

$$\min \left(\frac{\partial u}{\partial t} + H(t, x, u, \nabla u), u - g(t, x) \right) = 0. \quad (6)$$

where $g(t, x)$ is defined by the user. In particular it forces to have $u(t, x) \geq g(t, x)$ in the case of (6).

Also, the following obstacle equations can be considered :

$$\max \left(\frac{\partial u}{\partial t} + H(x, \nabla u), u - \tilde{g}(t, x) \right) = 0 \quad (7)$$

or

$$\max \left(\min \left(\frac{\partial u}{\partial t} + H(x, \nabla u), u - g(t, x) \right), u - \tilde{g}(t, x) \right) = 0, \quad (8)$$

where $\tilde{g}(t, x)$ is user-defined. We will have $u(t, x) \leq \tilde{g}(t, x)$, in the case of (7), or $u(t, x) \in [g(t, x), \tilde{g}(t, x)]$ in the case of (8).

For this, the following parameters are used :

- `OBSTACLE` $\in \{0, 1\}$
 - 0 : no obstacle g taken into account.
 - 1 : Equation (6) is treated, with the obstacle g .
- function `g_obstacle`
- `OBSTACLE_TILDE` $\in \{0, 1\}$
 - 0 : no obstacle $\tilde{g}$ taken into account.
 - 1 : Equation (7) is treated, with the obstacle $\tilde{g}$.
- function `g_obstacle_tilde`

4. Or `OUTPUT/VF_PROCxx.dat` values if parallel MPI is used.

- In the case we set `OBSTACLE=1` and `OBSTACLE_TILDE=1` then (8) is treated, and both functions g and $\tilde{g}$ must be defined. (The obstacle functions should satisfy $g(x) \leq \tilde{g}(x)$ in order to avoid undesired results).

Output parameters (output files are generally contained in the directory `OUTPUT/`)

- `COMPUTE_MAIN_LOOP` :
 - 1 : the main computational loop (iterative scheme) is called
 - 0 : the main loop is not called, only data initializations and savings are performed.
- `COMPUTE_VEX` $\in \{0, 1\}$: will save a `VEX.dat` file.
 - 1 : if the exact solution is given by the user (using function `Vex`)
 - 0 : no saving.

In particular it will compute errors (see below) relatively to this value.
- `COMPUTE_TMIN` $\in \{0, 1\}$: will compute file `tmin.dat` which contains the first time $t_n \in [0, T]$ (or some $P1$ interpolant) such that $v(t_n, x) \leq 0$ (Set the value to `INF=1.e5` otherwise.)
- `PRECOMPUTE_COORD` :
 - 1 : precomputes the coordinates : faster computations but more memory demanding.
 - 0 : no precomputations.
- `SAVE_VFALL` $\in \{0, 1\}$: to save the value of v each `SAVE_VFALL_STEP` iterations (into files `VFxx.dat`).
- `FORMAT_FULldata` $\in \{0, 1\}$: if set to 0, to save only the value at grid mesh points for files such as `VF.dat`
- `CHECK_ERROR` $\in \{0, 1\}$: to compute error every `CHECK_ERROR_STEP` steps (L^∞ and L^1 error computations) Errors are relative to the `Vex` function. Error are computed only in the region of points x such that $|\text{Vex}(\mathbf{t}, \mathbf{x})| < \text{C_THRESHOLD}$ ⁵
- `COUPE_DIMS` `[DIM]`, `COUPE_VALS` `[DIM]` :
 - list of integers n_i (0 or 1) to define the two variables used for the cut.
 - list of values (c_i or 0.) to define the position of the cut. The values c_i are used only when $n_i = 0$.
 - The output is in the file `output.dat`
 - Example : `COUPE_DIMS` `[DIM]` `= {1, 0, 1}` and `COUPE_VALS` `[DIM]` `= {0., 0.5, 0.}` define a cut in the plane $x_2 = 0.5$.

Parameters for trajectory reconstruction The dynamics used is the one defined in `dynamics`.

5. For the moment, `C_THRESHOLD` is defined in `src/main.cpp`.

The following parameters are for control discretization (they are similar to the one used in the case `COMMANDS=1` or `COMMANDS=2`).

- `CDIM` : dimension of the control (default is 2)
- `NC` : total number of commands
- `NCD[CDIM]` : tab indicating the number of discretisation points for each control `u[i]`.
- `UMIN`, `UMAX` : bounds for each control parameter

Then the following more specific parameters are used :

- `TRAJPT` : $\in \{0,1\}$: set to 1 for trajectory reconstruction, 0 otherwise.
- `initialpoint[DIM]` : coordinate of initial point x_0 for trajectory reconstruction.
- `TRAJ_METHOD` : method of reconstruction : 0 if based on minimal time (utilizes `tmin.dat`), 1 if based on the value function (utilizes `VFxx.dat` files).
- `time_TRAJ_START` : starting time t_0 used in the case `TRAJ_METHOD=1`. Can be any value in $[0, T[$. The program will construct an approximated trajectory such that $\dot{y}(t) = f(t, y(t), \alpha(t))$, $t \in [t_0, T]$, and starting with $y(t_0) = x_0$ as initial point.⁶
- Note that several trajectory reconstructions can be obtained by setting `TRAJPT` ≥ 1 , equal to the desired number of trajectories, and then `initialpoint[TRAJPT*DIM]` should contain list of corresponding points.
- If `TRAJPT=0` then one should define accordingly `initialpoint[DIM]={}`.

4 Output files

Output files are generally contained in the directory `OUTPUT/`

- `OUTPUT/VF.dat` contains the final u at the end of the computation. The file is structured as follows, on each line (case `FORMAT_FULldata=1`) :

`i1 i2 .... id val`

where `val` corresponds to the value $u(T, x)$ at mesh point $x = (x_{i_1}, \dots, x_{i_d})$.

In the case `FORMAT_FULldata=0`, on each line, in the same order, only the value

`val`

will be present in the file.

- `OUTPUT/VEX.dat` contains the final value as programmed in `Vex` (if `COMPUTE_VEX=1`).
- `OUTPUT/coupe.dat` (and `coupeex.dat`) can be obtained in the same way, for a problem in dimension $d \geq 3$ and when we make a "cut" in some particular subspace with some given coordinates, and that we want to visualize the results. See paragraph "Output parameters" for using `COUPE_DIMS` and `COUPE_VAL`. This is typically used to make a 2d cut (see for instance example `data/data_FD_3d.h`).

6. if $\bar{V}(t_n, x)$ is the value function at time t_n , attempt to reconstruct a trajectory such that $\inf_a \bar{V}(t_{n+1}, y_{t_n, x}^a(t_{n+1}))$ is minimal (this in the case `OPTIM=MAXIMUM`).

Furthermore, the following files are related to [trajectory reconstruction](#) :

- `OUTPUT/traj-n.dat` : in case $\text{TRAJPT} \geq 1$, corresponds to the trajectory number n . Each lines has the form

`x1 x2 ... xd a1 ... ap t`

where x_i are the coordinates of the point at time t , and a_j the corresponding control parameters.

- `OUTPUT/successTrajectories.dat` Each line n of this file has the form

`x1 x2 ... xd t success`

and indicates whether the trajectory number n was successful or not (`success=1` or `0`) and shows the corresponding terminal coordinates (x_i) and time t of the last constructed point.

5 Graphic outputs

Matlab : the file `OUTPUT/output_view.m` can be executed by typing '`output_view`' in the Matlab's user interface (we advise to first set the current Matlab's directory to `OUTPUT/`). some parameters are given below :

For the first figure :

- `level_set` : a level set value.
- `PLOT_CONTOUR` $\in \{0,1\}$: if set to 1, will plot contours of the output with `level_set` value.
- `PLOT_REACHABLE` $\in \{0,1\}$: if set to 1, will plot the 2d set of points where output value is $\leq \text{level_set}$.

This plot is based on the output value which is in the file `VF.dat` for $\text{DIM}=2$, and a priori in the file `coupe.dat` for $\text{DIM}>2$.

For the second figure :

- `PLOT_3d` $\in \{0,1\}$: set this parameter to 1 in order to obtain a 3d graph of the result.
- `PLOT_TMIN` $\in \{0,1\}$: (default is 0). Set this parameter to 1 in order to obtain a 3d graph of the minimal time (to reach a target). Will then use output `tmin.dat`, and will draw several contour plots ranging from 0 to T (plotting `tmin.dat` is only ok for $\text{DIM}=2$).

Also for both figures,

- `AXIS_EQUAL` $\in \{0,1\}$, if set to 1, will make "axis equal",
- `TRAJECTORY` $\in \{0,1\}$, if set to 1, will line-plot the list of first two components (x_1, x_2) of file "trajectory.dat".

Paraview : For the moment this option is only working for some 2d/3d cases.
Launch "paraview", then load the files in `VTK/*` (load `tab*` and so on).

6 Examples

Rotation example : We give two ways to program an advection-like example, in files `data_basicmodel.h` and `data_advancedmodel.h`. We consider the equation

$$\partial_t u + \max(0, -f(x) \cdot \nabla u) = 0.$$

with the parameters $\Omega = [-2, 2]^2$, $T = 0.5$, $f(x_1, x_2) = 2\pi(-x_2, x_1)$. (Hence the Hamiltonian $H(x, p, t) = \max(0, -f(x) \cdot p)$.)

In the first way (`data_basicmodel.h`), we define dynamics

$$f(x, a) = af(x)$$

with two control values $a \in \{0, 1\}$.

In the second way (`data_advancedmodel.h`), we define the Lax-Friedrich numerical hamiltonian H_{num} associated to H (see (11)).

Eikonal equation : see the files `data_FD_2d_ex1_basic.h` and `data_FD_2d_ex1_advanced.h`. The problem solved is

$$\partial_t u + c(x, t) \|\nabla u\| = 0, \quad x \in [-2, 2]^d, \quad t \in [0, T]. \quad (9)$$

$$u(0, x) = u_0(x), \quad x \in [-2, 2]^d, \quad (10)$$

with $d = 2$, $T = 1$, here $c(x, t) \equiv 1$, and with some (radially symmetric) initial data $u_0(x)$.

In the file `data_FD_2d_ex1_basic.h`, a scheme is programmed using a control-discretisation of $\|\nabla u\|$ as follows :

$$\|\nabla u\| \sim \max_{k=1, \dots, \text{NCD}} \langle (\cos(\theta_k), \sin(\theta_k), \nabla u), \quad \theta_k = \frac{2k\pi}{\text{NCD}}$$

where NCD is the number of controls.

In the file `data_FD_2d_ex1_advanced.h`, a scheme is programmed using a Lax-Friedrich numerical approximation H_{num} associated to H :

$$H_{num}(x, (p_1^-, p_1^+), \dots, (p_d^-, p_d^+), t) := H(x, \frac{p^- + p^+}{2}, t) - \sum_{i=1}^d c_i \left(\frac{p_i^+ - p_i^-}{2} \right) \quad (11)$$

The exact solution is given for comparison.

7 Second order HJB equations and SL scheme

The problem solved is the following second order Hamilton-Jacobi (HJ) equation

$$\frac{\partial u}{\partial t} + \lambda(x)u + \quad (12a)$$

$$\max_{a \in \mathcal{A}} \left(-\frac{1}{2} \text{Tr}(\Sigma(t, x, a) \Sigma^T(t, x, a) D^2 u) - B(t, x, a) \cdot \nabla u + r(t, x, a)u - \ell(t, x, a) \right) = 0$$

$$t \in (0, T), \quad x \in \mathbb{R}^d,$$

$$u(0, x) = u_0(x), \quad x \in \mathbb{R}^d \quad (12b)$$

where $\mathcal{A}$ is some non empty compact subset of $\mathbb{R}^m$ ($m \geq 1$), $B(t, x, a)$ is a vector of $\mathbb{R}^d$, $r(t, x, a)$, $\ell(t, x, a)$, are real-valued, and $\Sigma(t, x, a)$ is a $d \times p$ real matrix (for some $p \geq 1$). This problem is linked to the computation of the value function of stochastic optimal control problems.

It is also possible to consider the corresponding steady equation, that is, equation (12a) alone, with no term $\frac{\partial u}{\partial t}$.

The c++ proposed solver is based on an SL method.⁷

User inputs :

- ORDER = 2
- PARAMP (denoted p below) This is in general set to $p = d$ in the scheme.
- function **Sigma** : $\Sigma(x, k, a, t)$ is the k th column vector of the matrix $(\Sigma_{ik}(t, x, a))$
- function **Drift** : $B(x, k, a, t)$, corresponding to a set a set of p vectors $B^k(t, x, a)$ ($k = 1, \dots, p$) of $\mathbb{R}^d$ such that

$$\sum_{k=1, \dots, p} B^k(t, x, a) = B(t, x, a).$$

(For instance the user can set $B^1(t, x, a) = B(t, x, a)$ and $B^k = 0, \forall k \geq 2$; or $B^k =$ the k -th component of B , the other components beeing set to zero, if $p = d$).

- function **funcR** : $r(t, x, a)$
- function **distributed_cost**(**x,t,x**) : distributive cost $\ell(t, x, a)$.
- function **funcL**(**x**) : $\lambda(x)$ (assuming $\lambda(x) > r(t, x, a)$ in the case of a steady equation, that is, equation (12a) alone, with no term $\frac{\partial u}{\partial t}$).

Scheme definition : we consider the following SL scheme, implemented on the points x of the grid $\mathcal{G}$. The initialization is done by

$$v^0(x) = u_0(x), \quad x \in \mathcal{G}.$$

7. For advanced users : it is programmed in function "secondorder_itSL_evo" that might be available in the source file HJB_SL.cpp, or in /include/secondorder_itSL_evo.h.

For $n = 0, \dots, N_T - 1$ (time iterations) (or untill some stopping criteria is satisfied in the case of steady equations), for all grid points $x \in \mathcal{G}$, we consider

$$v^{n+1}(x) := \min_{a \in \mathcal{A}} \frac{e^{-r(t,x,a)\Delta t}}{1 + \lambda(x)\Delta t} \left(\frac{1}{2p} \sum_{\substack{k=1, \dots, p \\ \epsilon=-1, 1}} [v^n](y_x^{k,\epsilon,u}(\Delta t)) + \Delta t \ell(t, x, a) \right). \quad (13)$$

where the "characteristics" $y_x(h)$ can be defined, at iteration $t = t_n$, for some $h \geq 0$, for instance by :

$$y_x^{k,\epsilon,a}(h) := x + B^k(t, x, a)h + \epsilon \Sigma^k(t, x, a)\sqrt{h}, \quad \epsilon \in \{-1, 1\}, \quad k = 1, \dots, p. \quad (14)$$

Also, $[v^n](y)$ denotes some interpolation of v^n at point y (typically Q_1).⁸

8 Source architecture

In this section, we present the architecture used in the implementation of the project. The `HJB` class represents the main class and the `DF` and `SL` classes are sub-classes of `HJB`. `Commands` and `Mesh` are separated classes. We remind that the user needs to code the problem parameters into a C++ header file and to include its name in the file `data/data_simulation.h`. For example, if the user data file is called `data_myexample.h`, then the line `#include "data_myexample.h"` (and only this one) must be included in the file `data/data_simulation.h`.

8. When using the definition (14), the scheme is of expected order $O(\Delta t) + O(\frac{\epsilon}{\Delta t})$ where ϵ is of the order of the interpolation error $\|v^n - [v^n]\|_\infty$. For smooth data, this interpolation error is of order Δx^2 for Q^1 interpolation, where Δx is the spatial mesh size.

